Distributed Systems

CS425/ECE428

March 22 2023

Instructor: Radhika Mittal

Acknowledgements for the materials: Nikita Borisov
Logistics

• Please refrain from discussing anything about the midterm with others after you have taken your exam.

• Final exams: May 4\textsuperscript{th} to May 11\textsuperscript{th}.
  • You can start making reservations on PrairieTest.

• Thank you for your feedback for those who have filled it up.
  • For those who haven't, please fill it up by Friday this week.
Agenda for today

• Consensus
  • Consensus in synchronous systems
    • Chapter 15.4
  • Impossibility of consensus in asynchronous systems
    • We will not cover the proof in details
  • Good enough consensus algorithm for asynchronous systems:
    • Paxos made simple, Leslie Lamport, 2001
• Other forms of consensus algorithm
  • Raft (log-based consensus)
    • Block-chains / Bitcoins (distributed consensus)
Bitcoins

- Implement a *distributed* replicated state machine that maintains an *account ledger* (= *bank*).
  - No user should be able to “double-spend”.
  - Need to know of all transactions to validate this.
  - Who does this validation? Cannot trust a single central authority.
    - Any participant (replica) should be able to validate.
    - All replicas must agree on the single history on transaction ordering.
- Scale to thousands of replicas distributed across the world.
- Allow old replicas to fail, new replicas to join seamlessly.
- Withstand various types of attacks.
Uses Blockchains for Consensus

• Why not use Paxos / Raft?
  • Need to scale to thousands of replicas across the world.
  • May not even know of all replicas a priori.
  • Participants may leave / join dynamically.
  • Paxos/Raft are ill-suited for such a setup.
    • Leader election in Raft or proposals in Paxos require communication with at least a majority of servers.
    • Require knowing the number of replicas.
    • ....

• So how does blockchain work?
  • Focus of today’s class. Only a high-level discussion.
Basic Idea

Transactions grouped into a *block* that gets added to the *chain* (history of transactions) by the “leader of that block”.
Lottery Leader Election

• Every node chooses a random number

• Leader = “closest to 0”
Lottery Leader Election

• Every node chooses a random number
  • The method for choosing the number in blockchains enables log consensus (with a high probability).
  • Requires the leader to expend CPU (as proof-of-work).

• Leader = “closest to 0”
  • Defined such that a replica can determine this independently without coordination
Choosing the random number

• Cryptographic hash function:
  • \( H(x) \rightarrow \{ 0, 1, \ldots, 2^{256} - 1 \} \)

• Hard to invert:
  • Given \( y \), find \( x \) such that \( H(x) = y \)
  • E.g., SHA256, SHA3, ...

• Every node picks random number \( x \) and computes \( H(x) \)

• Node with \( H(x) \) “closest to 0” wins
  • Finding such an \( x \) requires expending CPU (proof-of-work).

• But once we have found an ‘\( x \)’, we can always be the leader for all blocks, or even share it with colluding parties. How to prevent that?
Using a seed

• Every node picks $x$, computes $H(\text{seed} \ || \ x)$
• Closest to 0 wins
• What to use as a seed?
• Hash of:
  • Previous log
  • Node identifier
  • New messages to add to log

• How to find “closest to 0”?
Iterated Hashing / Proof of work

• Repeat:
  • Pick random x, compute $y = H(\text{seed} \ || \ x)$
  • If $y < T$, you win!

• Set threshold $T$ so that on average, one winner every few minutes

• Given a solution, $x$ such that $H(\text{seed} \ || \ x) < T$, anyone can verify the solution in constant time (microseconds).
Chaining the blocks

Alice generated 50 BTC
Nonce: 1234

Bob generated 50 BTC
Nonce: 5678

Carol generated 50 BTC
Alice transferred 10 BTC to Bob + 1 BTC to Carol (fee)
Nonce: 9932

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>39 BTC</td>
</tr>
<tr>
<td>Bob</td>
<td>60 BTC</td>
</tr>
<tr>
<td>Carol</td>
<td>51 BTC</td>
</tr>
</tbody>
</table>
Protocol Overview

• New transactions broadcast to all nodes.
• Each node collects new transactions into a block.
• Each node works on finding a proof-of-work for its block to become its leader and get it appended to a chain.
  • i.e. finds $x$, such that $H(\text{seed} \ || \ x) < T$.
• When a node finds a proof-of-work, it broadcasts it to all nodes.
• Nodes accept a block only if all transactions in it are valid.
• Nodes express their acceptance by working on creating the next block in the chain, using the hash of accepted block as previous hash.
What could go wrong?

• Two nodes may end up mining different versions of the next block.
• A node may receive two versions of the next block.
Two nodes may end up mining different versions of the next block.
A node may receive two versions of the next block.
Will store both, but work on the first one they receive.
Over time, more blocks will be received.
The node will switch to working on the *longest chain*.
When is a transaction committed?

- Wait for up to $k$ more blocks to be added in the chain.
- Then commit the transaction.
- $k$ is set to 6 for Bitcoins.
Security Property

- Majority decision is represented by the longest chain.
  - It has greatest “proof-of-work” invested in it.

- If majority of CPU power is controlled by honest nodes, the honest chain will grow fastest and outpace competing chains.

- To modify past blocks, an attacker will need to redo the proof-of-work for that block, and all blocks after it, and then surpass the work of honest nodes.

- Probability of attack reduces as more blocks get added.
Incentives for Logging

• Security better if more people participated in logging.

• Incentivize users to log others’ transactions
  • Transaction fees: e.g. pay me x% to log your data (or some fixed fee per transaction)
  • Mining reward: each block creates bitcoins
Logging Speed

• How to set T?
  • Too small: slows down transactions
  • Too big: wasted effort due to chain splits

• Periodically adjust difficulty T such that one block gets added every 10 minutes.
  • Depends on hardware speed (which typically improves over time) and number of participants (which vary over time).

• Determined algorithmically based on the rate at which blocks are mined
  • Target is 1 block every 10 minutes.
  • Difficulty recomputed after every 2016 blocks.
Bitcoin Broadcast

- Need to broadcast:
  - Transactions to all nodes, so they can be included in a block.
  - New blocks to all nodes, so that they can switch to longest chain.

- What if we use R-multicast?
  - Have to send $O(N)$ messages
  - Have to know which nodes to send to
  - Not a suitable choice.
Each node connects to a small set of neighbors (10–100).
Nodes propagate transactions and blocks to neighbors.
Push method: when you hear a new tx/block, resend them to all (some) of your neighbors (flooding).
Pull method: periodically poll neighbors for list of blocks/tx’s, then request any you are missing.
Unreliable: some nodes may not receive all transactions or all blocks. But that’s ok.
Maintaining Neighbors

- A seed service
  - Gives out a list of random or well-connected nodes
  - E.g., seed.bitnodes.io

- Neighbor discovery
  - Ask neighbors about their neighbors
  - Randomly connect to some of them
Bitcoin Summary

• Unreliable broadcast using gossip
• Probabilistic “leader” election for mining blocks (tx ordering)
• Longest chain rule to ensure long-term (probabilistic) consistency and security

• Compared with Paxos/Raft:
  • Scales to thousands of participants, dynamic groups
  • Tens of minutes to successfully log a transaction (vs. milliseconds)